CarbonShare: Carbon-Fair Allocation for Shared Clusters

John Thiede
University of Massachusetts Amherst
Amherst, Massachusetts, USA

David Irwin
University of Massachusetts Amherst
Amherst, Massachusetts, USA

Prashant Shenoy
University of Massachusetts Amherst
Amherst, Massachusetts, USA

Abstract

Computing's energy demand, and thus its carbon emissions, are rapidly accelerating with the emergence of a wide range of useful, but computationally-intensive, AI-driven applications. At the same time, computing, along with the rest of society, must rapidly reduce its emissions to avoid the worst consequences of climate change, e.g., population displacement, agricultural collapse, mass extinction, extreme weather, etc. To do so, computing and other industries will eventually need to limit their carbon emissions. Such a limit imposes a new allocation problem: how should datacenters allocate their limited carbon emissions across multiple applications?

To address the problem, we introduce the notion of carbon fairness, which enables clusters to enforce a limit on their aggregate carbon emissions while fairly allocating them to applications. As we show, enforcing carbon fairness has multiple desirable properties: it fairly distributes any performance penalty from a cluster-wide carbon cap across all applications; it incentivizes applications to optimize their energy- and carbon-efficiency; and it is scheduler agnostic by simply limiting the usage of allocated resources. We implement and evaluate a range of carbon fairness policies in LXD, and show i) how these policies differ from enforcing resource and energy fairness and ii) the importance of enabling flexible policies, which permit periodic bursts above carbon limits, to maintain performance despite limits on carbon. For example, we show that our most flexible *footprint-fair policy* outperforms a stricter *rate-fair policy* on a large-scale industry workload by up to 30%.

CCS Concepts

• **Hardware → Impact on the environment**; • **General and reference → Performance**; **Metrics**; **Design**.

Keywords

Carbon-efficiency, energy-efficiency, performance, datacenters.

ACM Reference Format:
John Thiede, David Irwin, and Prashant Shenoy. 2026. CarbonShare: Carbon-Fair Allocation for Shared Clusters. In *Proceedings of the 17th ACM/SPEC International Conference on Performance Engineering (ICPE '26), May 4–8, 2026, Florence, Italy.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3777884.3796993

ICPE '26, Florence, Italy.

ACM ISBN 979-8-4007-2325-4/2026/05
https://doi.org/10.1145/3777884.3796993

1 Introduction

Computing's energy demand is rapidly accelerating with the emergence of a wide range of useful, but computationally-intensive, AI-driven applications. Indeed, recent estimates project that datacenter energy demand will increase by 1.5-2× within the next five years [2, 3]. There is growing concern that this rapid increase in datacenter energy consumption combined with the electrification of transportation (via electric vehicles) and heating/cooling (via electric heat pumps) may exceed the grid's capacity, leading to higher energy prices as well as more frequent energy outages and shortages [44]. Even more alarming, to satisfy growing datacenter demand, many utilities are not only delaying the shutdown of carbon-intensive coal plants, but also adding more carbon-intensive generation sources [10, 17, 32, 35]. This has led technology companies' carbon emissions to increase significantly, causing them to miss their carbon reduction targets by large margins [11].

At the same time, computing, along with the rest of society, must rapidly reduce its emissions to avoid the worst consequences of climate change, such as large-scale population displacement, agricultural collapse, mass extinction, and increasingly extreme weather events. Importantly, grid energy's carbon-intensity varies significantly both spatially and temporally due to the mix of generation sources at any given location and time [39]. For example, during periods of low energy demand and sunny weather, grid energy's carbon-intensity may be quite low if it derives from a combination of hydro and nuclear baseload generators plus solar. In contrast, during periods of high energy demand and cloudy weather, grid energy's carbon-intensity may be much higher due to the heavy use of natural gas-powered peaking generators and lack of solar. Datacenters may also supplement grid energy with other low-carbon energy from solar, wind, and nuclear—either behind-the-meter or individually contracted through power purchase agreements—that reduces their aggregate energy supply's average carbon-intensity.

Prior work has analyzed and leveraged spatiotemporal variations in energy's carbon-intensity to optimize computing's carbon emissions by both i) dynamically shifting workloads across space and time [39, 43] and ii) adaptively scaling workloads' resource and energy usage in response to changes in carbon-intensity [25, 38]. Practical adoption of such methods will be driven by two trends. First, many local, state, and national governments are starting to regulate carbon emissions in some form [4], which will require datacenters and applications to also control and optimize their carbon footprint. We expect such regulations to increase and strengthen over time, providing an strong incentive to limit carbon emissions. Second, many companies have adopted sustainability goals to reduce their emissions over the next decade, which implies that their IT operations will also need to limit their carbon emissions.

As a result, recent work enables carbon monitoring and control for computing by combining fine-grained energy monitoring and control of processes and containers, e.g., via hardware counters and

resources quotas [34, 38, 40], with data from carbon information services, such as electricityMaps [5], which expose APIs to track grid energy's carbon-intensity at any location.

Ultimately, carbon regulation policies will impose a limit on datacenter carbon emissions over some interval, e.g., a month or year. Such a limit imposes a new allocation problem: how should datacenters allocate limited carbon across competing applications? As we discuss, while carbon allocation is related to resource and energy allocation there are important differences. In particular, a carbon allocation should "fairly" distribute the burden, and potential performance penalty, of limited carbon across applications, while also incentivizing them to conserve carbon by adopting the carbon optimizations above. To address the problem, we i) introduce the notion of carbon fairness, which enables clusters to enforce a limit on their aggregate carbon emissions while fairly allocating them to applications, ii) describe how it differs from resource and energy fairness, and iii) define and compare multiple carbon-fair policy variants for batch and interactive workloads. Specifically, our carbon-fair policies reveal a fundamental trade-off between enabling applications flexibility to satisfy variable demand—by using more carbon when demand is high and less when it is low—and increasing the risk that applications exhaust their carbon allocation too early. As we show, more flexible policies yield higher performance assuming accurate predictions of future resource demand and carbon emissions.

To evaluate our policies, we design a CarbonShare testbed that extends CloudLab to enable experimentation with energy- and carbon-aware software development. In particular, our testbed instruments datacenter power distribution units (PDUs) and servers' IPMI-enabled baseboard management controllers (BMCs) to monitor fine-grained server-level power consumption, temperature, fan speed, etc., and uses the electricityMaps API [5] to monitor the local grid's carbon-intensity. Our testbed makes this server-level monitoring data available programmatically in real-time via an API to our CarbonShare prototype. The testbed also includes a web portal for viewing and retrieving data archives. Importantly, our CarbonShare testbed operates in parallel with CloudLab, and does not require direct integration with it, as it uses only out-of-band management interfaces. This enables users to allocate CloudLab servers from a CarbonShare-enabled site, and then separately use the API to retrieve monitoring data for their specific servers.

Our hypothesis is that a *footprint-fair* carbon enforcement policy, which only limits users once they have exhausted their budget, provides applications maximum flexibility to burst above carbon limits under high demand and outperforms stricter *rate-fair* policies, which limit carbon emissions to a specified rate at all times, by enabling higher performance for lower carbon emissions. In evaluating our hypothesis, we make the following contributions.

Introduce Carbon Fairness. We introduce the notion of carbon fairness for clusters. Put simply, given a cap, or limit L, on clusters carbon emissions over some time interval T, a carbon-fair allocation policy ensures the sum of carbon emissions of individual applications does not exceed L, and that each application j's emissions $FairShare(j)$ are in proportion to their assigned carbon weight w_j.

Carbon-Fair Allocation Policies. We identify desirable properties for carbon-fair allocation, and then design *rate-fair* and *footprint-fair* variants that offer different tradeoffs between flexibility and risk. We analyze the extent to which these policies have the desirable properties above, and also how they compare with prior resource and energy fair allocation. Importantly, as we discuss, our carbon-fair allocation policies are scheduler-agnostic in that they do not require altering current resource schedulers and allocation policies, but only limiting the usage of allocated resources.

Implementation and Evaluation. We implement a CarbonShare prototype by extending the LXD container manager, and integrate it with our CarbonShare testbed, which provides API access to power consumption and carbon data for CloudLab servers. We then evaluate CarbonShare on CloudLab using industry traces for both batch jobs and interactive services. We show that our most flexible *footprint-fair* carbon allocation policy outperforms a stricter *rate-fair* policy by up to 30% more performance for similar carbon, and a carbon-agnostic policy by up to 35% less carbon for similar performance. Our results also highlight that carbon-fair allocation highly incentivizes carbon optimization at the application level.

2 Motivation and Background

Below, we provide background on grid energy's variable carbon-intensity, application-level energy and carbon attribution and control, and prior work that enforces cluster-level resource and energy fairness and how it differs from carbon fairness.

2.1 Energy's Carbon-Intensity

Energy's carbon-intensity (in $g{\cdot}CO_2e/Wh$) is defined as the amount of carbon equivalent emissions, i.e., in grams, per unit of energy produced, i.e., in watt-hours. Grid energy's carbon-intensity varies over time based on the mix of generators, e.g., nuclear, coal, oil, natural gas, solar, wind, hydro, etc., producing the energy and their carbon-intensity. Specifically, energy's average carbon-intensity at any time is based on the average of the carbon-intensity for each generation source weighted by the energy it is producing. The mix of generators, and the amount of energy they generate, varies based on the energy demand and, in the case of renewables, the environment. That is, as demand rises or renewable generation drops, the grid must activate additional, often dirtier, generators to satisfy the demand. Since the grid is divided into many balancing authorities, which must satisfy their own local energy demand with their own mix of generation sources, energy's carbon-intensity also varies widely across regions. Carbon information services, such as electricityMaps [5], leverage publicly-available grid generation data to estimate real-time grid carbon-intensity for numerous regions worldwide and make it available in real time via web APIs.

Figure 1 depicts an example showing variations in grid energy's carbon-intensity over a 4-day period for three different regions according to electricityMaps. As shown, these regions have widely different magnitudes and variations in their energy's carbon-intensity based on their generation mix. Specifically, California and Australia exhibit higher and more variable carbon due to their mix of both solar energy and fossil fuels, while Norway has much lower and less variable carbon due to its heavy use of hydroelectric power.

Note that we focus on fairly allocating a fixed carbon budget at a single datacenter, and not on migrating workloads across regions. However, as we discuss, an important goal of CarbonShare is to

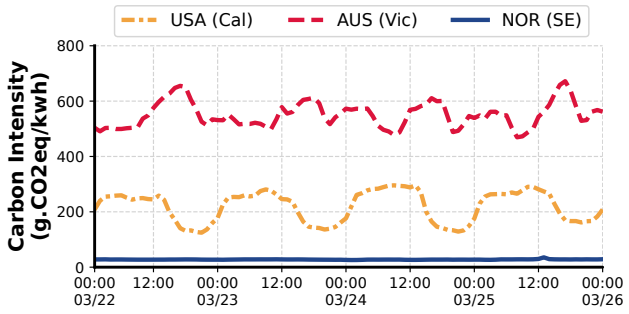

Figure 1: ***Grid energy's variable carbon-intensity across three different regions over a 96-hour period.***

introduce an incentive for applications to optimize their carbon-efficiency. Such an incentive could encourage migration to lower carbon regions, although evaluating such benefits are outside our scope. We instead focus on the benefits to carbon-efficiency of time-shifting workloads in regions where carbon-intensity varies. Even in regions where carbon-intensity does not vary, CarbonShare still enables datacenter operators to limit their emissions.

2.2 Carbon Attribution and Control

Given the carbon-intensity data for a region above, carbon emissions for a server (in g·CO_2e) over any interval T is simply the product of its energy consumption over that interval and its energy's average carbon-intensity. Server energy usage is typically metered and can be externally monitored using power distribution units (PDUs) or baseboard management controllers (BMCs) via various protocols, such as IPMI. In general, server power consumption has both a baseload and marginal component. The baseload component represents the power draw of the server when idle, while the marginal component is the additional power the server consumes when utilized. In general, a server's marginal power is roughly proportional to its core utilization at any fixed frequency. In addition, when using dynamic voltage and frequency scaling (DVFS), the marginal power at a fixed utilization varies roughly linearly with changes in frequency and quadratically (i.e., as the square of) voltage. GPUs exhibit similar baseload and marginal power characteristics, and can also be externally monitored, e.g., using command-line tools like `nvidia-smi`.

Importantly, though, CarbonShare requires attributing power and carbon on a per-application basis, and unlike servers and GPUs, individual applications running on the same server cannot be physically metered. Since marginal power consumption is a function of resource usage, prior work, such as power containers [37] and PowerAPI [13], has addressed this problem by monitoring per-application resource usage and then developing models that map their fraction of server-level resource usage to their fraction of server-level power consumption. These models generally monitor a range of hardware counters (beyond simple core utilization) to infer and attribute per-application power usage. Recent work, such as carbond [34] and carbon containers [40], has extended this basic model-based approach to monitoring per-application carbon by combining individual applications' estimated power with energy's carbon-intensity. CarbonShare builds on this prior work, and takes a similar model-based in its prototype.

In addition to attributing power and carbon to applications, CarbonShare also requires mechanisms for enforcing limits on both power and carbon. To do so, CarbonShare leverages prior work on power containers and carbon containers, which both use the power models above to set resource quotas, e.g., on core utilization, memory, etc., on per-application containers to enforce a strict upper limit on their power consumption and carbon emissions. While these mechanisms are per-container for individual applications, CarbonShare coordinates these limits across multiple applications and containers in a cluster based on its allocation policy, which we discuss in the next section. As in prior work [28], CarbonShare does not use hardware-specific mechanisms, such as RAPL and DVFS, for controlling power, as these mechanisms are not generally available on all servers and require more sophisticated power models.

2.3 Fair-share Allocation

While there has been extensive prior work on fair-share resource and energy allocation [20, 21, 29, 42], it does not directly apply to carbon fairness. To understand why, consider that while Dominant Resource Fairness (DRF), a common fair-share allocation policy for clusters, dictates the cores and memory allocated to each job, it does not limit their usage, and thus power consumption, based on energy's carbon-intensity. Thus, some jobs may operate their allocated resources at full utilization and high power during a high-carbon period, while others may operate them at lower utilization and thus lower power during a low-carbon period. Such behavior can create a significant discrepancy in jobs' carbon emissions for the same resource allocation. Similarly, similar cluster-level energy-fair allocation policies [20] implicitly assume that all energy is created equal and does not account for variations in energy's carbon-intensity over time. As a result, while these policies do account for resource and power usage of allocated resources, they do not consider when the energy is consumed and energy's carbon-intensity at that time. Thus, two jobs may have the same resource and power usage, but widely different carbon emissions based on when they consumed the power. For clusters with a limited aggregate carbon budget, unfairness with respect to carbon emissions is problematic, as it provides no incentive for jobs to limit their emissions by responding to potential variations in carbon emissions over time.

2.4 Problem Definition

CarbonShare addresses the following problem: given a limit on carbon emissions L that a cluster cannot exceed, the system must fairly budget this limit across a set of jobs (or applications) J while maximizing its computational work. CarbonShare differs from related work on Fair-CO2[24] that provides a framework for fair attribution of operational and embodied carbon emissions in a shared computing environment. Instead, CarbonShare focuses on allocating a limited operational carbon budget across multiple applications running on a shared cluster. We discuss our methodology for carbon attribution in section 4. A naive fair allocation would split the budget equally between the jobs. While this is intuitively "fair," jobs' differing demands may cause unfair outcomes.

Thus, we propose a carbon-fair attribution policy based on weighted fair division. For all jobs, CarbonShare assigns a proportional weight

that determines each job's share of the carbon budget. Each job is guaranteed to receive a portion of the budget that is proportional to their weight. The cluster administrator assigns weights based on a configurable policy. Since CarbonShare admits a wide range of weight assignment policies, exploring all such policies is outside the scope of this paper. For example, a weight assignment policy may incorporate factors not directly related to fairness or efficiency, e.g., giving higher weights to certain users or applications based on some priority. In this paper, we explore how specific design factors, such as enforcement methods and policies, affect cluster performance under a limited carbon budget. We also examine how factors, such as job behavior, carbon variance, and carbon forecasting uncertainty, may affect performance.

In this paper, we assume a policy that assigns weights to jobs based on their computing demand over a budgeting period. This assignment of weights represents the optimal policy in terms of fairness and performance, and is based on the notion of Shapley values [36] from cooperative game theory, which fairly distributes individual payoffs of a coalition of players participating in some form of collaborative game. Here, the game is to maximize the "value" of the collective under the given limit while appropriately assigning budgets to each player based on their contribution to the collective value. For the purposes of our evaluation, we assume that jobs are independent from each other and that each job's value is their potential computing work over a given budget period. In such a case where all participants in a game are independent, their Shapley values are equivalent to their individual value by definition. By setting these weights proportional to the amount of computing demand, jobs are given appropriately-sized budgets that avoid the disproportionate performance burdens mentioned above. We examine the effect of using different weight assignment policies that are simple approximations of this optimal solution in Section 5.2.

3 CarbonShare Design

CarbonShare is intended to be integrated with an existing cluster resource manager, such as Kubernetes [7], Borg [41], LXD [30], etc., that allocates resources to cluster applications in the form of containers. Our prototype extends LXD, as we discuss in Section 4. These systems allocate and manage resources for applications and users in the form of one or more containers, and already include sophisticated scheduling policies, such as DRF, to arbitrate access to limited hardware resources. We design CarbonShare to be scheduler-agnostic such that it works in conjunction with any scheduler and scheduling policy. In particular, CarbonShare does not alter cluster scheduling policies that typically allocate resources, e.g., cores and memory, to applications across one or more containers. However, once resources are allocated to applications, CarbonShare may restrict their resource and energy usage to ensure that an application does not exceed its carbon share based on its past emissions and grid energy's current carbon-intensity. Thus, managing carbon emissions requires periodically imposing limits on applications' usage of their allocated resources (and energy) but does not directly affect these allocations by the scheduler. That said, as we discuss, there is a relationship between applications' resource share and their carbon share that affects cluster-wide performance.

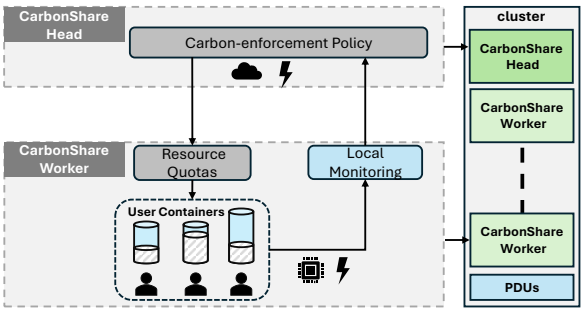

Figure 2: *Overview of CarbonShare design.*

Figure 2 illustrates CarbonShare's design, which consists of multiple coordinated microservices that facilitate i) per-container carbon monitoring and ii) the system-wide enforcement of carbon limits based on a fair-share allocation policy by leveraging container-level resource allocation and suspend/resume mechanisms.

Monitoring Carbon. CarbonShare includes a power monitoring subsystem that interfaces with servers' PDUs (or BMCs) to monitor per-server power, and make it available internally to CarbonShare via an API. CarbonShare receives real-time server-level power at a configurable interval, e.g., every second, via the API, and uses a power model, as discussed in the previous section, to attribute the marginal power to each server's hosted containers based on their resource usage. As we show in Section 4, marginal power is generally roughly linear to core utilization. Attributing baseload power to containers is a policy decision and can either be split equally among containers or in proportion to their resource usage. CarbonShare retrieves carbon-intensity data via electricityMaps' API to compute carbon emissions based on each container's power usage, and also tracks the aggregate emissions across all containers.

Enforcing Carbon Limits. CarbonShare uses the power model above to set resource quotas on a per-container basis to enforce an upper limit on container power usage. CarbonShare dynamically adjusts these resource quotas based on changes in energy's carbon-intensity to control the upper limit on a container's rate of carbon emissions. CarbonShare can also suspend containers that cannot satisfy their carbon limit via vertical scaling using resource quotas, i.e., if energy's carbon-intensity rises too high.

Policy Module. Finally, CarbonShare includes a policy module that enables administrators to assign weights to applications and tracks their carbon emissions via the monitoring data and enforces carbon-fair allocation by dynamically adjusting carbon limits. We discuss CarbonShare's carbon-fair allocation policies below.

3.1 Carbon-fair Allocation Policies

CarbonShare ensures that a cluster does not exceed a carbon emissions limit L over some time interval T, such as week or month. To do so, CarbonShare assigns a carbon allocation to each job (or user) j such that the aggregate carbon emissions for each job are in proportion to the job's weight (w_j), assuming the job consumes its entire carbon share. For a given job j, CarbonShare calculates its carbon allocation as:

$$FairShare(j) = \frac{w_j \cdot L}{\sum weights} \quad (1)$$

where $\sum weights$ is the sum of all job weights. A job's actual carbon emissions may be less than its assigned share if it does not use its full carbon allocation. As a result, CarbonShare also ensures that the sum of the carbon emissions across all jobs is less than or equal to the carbon emissions limit L. We assume a cluster's aggregate carbon limit L is set by some exogenous policy based on companies' internal carbon emissions targets or government regulations. Similarly, the cluster operator defines the policy that assigns weights to different applications. In general, providing a higher carbon share to more resource-intensive applications improves performance. We discuss different policies for setting the weights below, and evaluate them in §5. In the following, we discuss CarbonShare in the context of shares assigned to individual jobs.

Given the definition of carbon-fair allocation above there are multiple policies that can yield carbon fairness. As described below, we define these policies along two extremes that differ in the amount of risk they impose on applications of depleting their cumulative carbon allocation before the end of their budget period T.

Rate-fair Policy. A rate-fair carbon allocation policy imposes no risk on applications depleting their carbon, as it enforces that applications emit carbon at a fixed rate (in $g{\cdot}CO_2e$ per unit time) such that total emissions over the budgeting interval T are equal to applications' carbon share. With this policy, CarbonShare enforces a limit at each time t on applications' maximum resource usage to ensure that the product of their corresponding maximum energy usage and energy's carbon-intensity at time t is equal to the fixed rate. CarbonShare calculates this fair rate as:

$$FairRate(j, T) = \frac{FairShare(j)}{T} \tag{2}$$

CarbonShare varies applications' resource limit over time based on the maximum power draw per-resource and variations in energy's carbon-intensity, such that, at the same fixed rate, a lower carbon-intensity enables higher resource usage and a higher carbon-intensity requires lower resource usage. Thus, the rate-fair policy incentivizes applications to use their entire carbon allocation, as they have no flexibility to reduce their carbon rate at some times and increase their rate at other times.

Our carbon rate-fair policy is similar to prior fair-share resource allocation policies that prevents applications from accruing credit when not consuming resources, as doing so could enable applications to starve others by remaining idle for a significant period, accruing significant credit, and then using it to prevent other applications from executing. Indeed, fair-share resource allocation only ensures applications' resource usage is proportional to their assigned weights when applications are using the resource and not idle. In general, most fair-share schedulers are work-conserving in that the resource shares of idle applications are re-distributed to active applications in proportion to their weights. CarbonShare can configure its rate-fair policy to be work-conserving in the same way, such that if any application's carbon-rate is less than its fixed rate (based on its assigned carbon share) it is re-distributed to other applications in proportion to their carbon share.

The problem with a work-conserving scheduler in the context of CarbonShare is that a key goal of the system is to incentivize applications to reduce their carbon emissions, both by improving their energy-efficiency (i.e., in work per unit of energy consumed) and carbon-efficiency (i.e., doing more work when energy's carbon-intensity is low). Instead, a work-conserving scheduler incentivizes applications to always use resources and energy up to their fixed carbon rate even if that usage is unnecessary or inefficient, since they either use their carbon share or they lose it. That is, an application that operates below its fixed carbon rate may not reduce the cluster's carbon emissions, as some other application may consume that carbon. Thus, we also enable configuring CarbonShare to be "carbon-conserving" such that any unused carbon shares are not re-distributed to applications. In this case, applications that operate below their fixed carbon rate are contributing to reducing the carbon emissions of the entire cluster, and not simply transferring their carbon emissions to someone else.

Footprint-fair Policy. Our *footprint-fair* policy differs from the rate-fair policy by not imposing a fixed carbon emissions rate, but simply tracks each application's carbon emissions over the budget interval T, and halts the application once its carbon emissions reach its allocated carbon limit $FairShare(j)$. This policy provides applications maximal flexibility to shift their carbon emissions over the budget interval by imposing no limit on their carbon rate. While the rate-fair policy ensures applications never deplete their carbon budget early, its primary issue is that it is inflexible. This inflexibility is a problem for jointly optimizing performance and carbon for both interactive and batch applications.

For interactive applications, the rate-fair policy provides no flexibility to increase resource usage when demand is high to satisfy latency SLOs, even if energy's carbon-intensity is also high, and then "pay it back" later when demand and carbon-intensity may both be low. Thus, while it may be possible for an interactive application to satisfy its latency SLOs within its carbon allocation over the budget interval T based on the demand and carbon-intensity profile, the rate-fair policy may not enable the flexibility to do so. As a result, interactive applications under the rate-fair policy may experience periods of high latency above the SLO (when both demand and carbon-intensity are high) and low latency well below the SLO (when both demand and carbon-intensity are low). In contrast, a more flexible policy might be able to provide a consistent latency near the SLO at all times by increasing resource usage during high demand times at a high carbon rate and decreasing resource usage during low demand times to compensate.

Since batch applications are non-interactive, they generally have even more flexibility to shift and scale their resource usage over time to better align with low-carbon periods. However, the rate-fair policy also prevents batch applications from fully exercising their flexibility to reduce their carbon emissions. For example, assuming a batch application has perfect knowledge of future carbon-intensity, it would plan its resource usage to use its carbon share over the budget period by fully utilizing its resources during the lowest carbon times and idling at other high-carbon times. Enabling this flexibility is the goal of our footprint-fair carbon allocation policy. Since this policy does not restrict applications' resource usage before hitting their carbon limit, it provides applications maximal flexibility to shift their usage in time and adjust their carbon rate, but also imposes maximal risk that applications may exhaust their carbon share before the budget limit.

Thus, the footprint-fair policy requires applications to effectively plan their usage over time based on their expected demand and

energy's carbon-intensity, and can enable higher performance for lower carbon emissions than the rate-fair policy if expectations are accurate. In §5, we evaluate the sensitivity of the footprint-fair policy to the accuracy of carbon-intensity forecasts. The footprint-fair policy is inherently carbon-conserving, and not work-conserving, since its flexibility derives from applications effectively saving their carbon share for use later, and thus does not permit implicitly transferring any unused carbon share to another application.

3.2 CarbonShare Properties

CarbonShare satisfies a number of desirable general and carbon-specific fairness properties. CarbonShare trivially satisfies many general fairness properties, such as envy-freeness (applications with the same carbon share have no preference for the other's share) and pareto-efficiency (increasing the allocation of one user must decrease the allocation of another). Below, we enumerate a number of carbon-specific fairness properties that are also important.

- *Burden Sharing*. CarbonShare's primary purpose is to ensure burden sharing such that all applications share the burden, and potential performance penalty, of adhering to a cluster-wide carbon limit, and that no application should bear more of this burden than any other. Thus, CarbonShare distributes the carbon burden in proportion to the weights.
- *Carbon Optimization Incentive*. CarbonShare also introduces a carbon optimization incentive that encourages applications to align their resource usage with periods of low carbon emissions. Prior work has demonstrated that applications have multiple degrees of temporal, resource, and performance flexibility. For example, prior work has leveraged temporal flexibility by pausing jobs when carbon emissions exceed some threshold [43], resource flexibility by scaling elastic jobs' resource usage up and down as energy's carbon-intensity falls and rises [25], respectively, and performance flexibility by gracefully managing latency SLOs with a limited carbon budget [38]. CarbonShare incentivizes and enables applications to leverage this flexibility on a shared infrastructure.
- *Scheduler Agnostic*. CarbonShare's approach is scheduler-agnostic in that its carbon-fair allocation policy is orthogonal to resource allocation. As mentioned above, the scheduler allocates resources based on its own policy, such as DRF, while CarbonShare limits the usage of those resources to enforce the application's assigned carbon share. Thus, CarbonShare inherits the advantages of existing schedulers. Of course, the relationship between an application's resource share and its carbon share is important, as assigning a high resource share and low carbon share may force resources to be under-utilized. We evaluate this relationship in §5.
- *Carbon-conserving*. Finally, as mentioned above, CarbonShare supports carbon-conserving scheduling that represents the dual of work-conserving scheduling such that applications' unused carbon shares are not simply given to competing applications, but may instead be saved for use in the future or counted towards carbon reductions. CarbonShare avoids the starvation issue with enabling applications to accrue credit from idle resources by decoupling resource and carbon allocation in extending existing cluster resource schedulers that allocate fixed resources to applications.

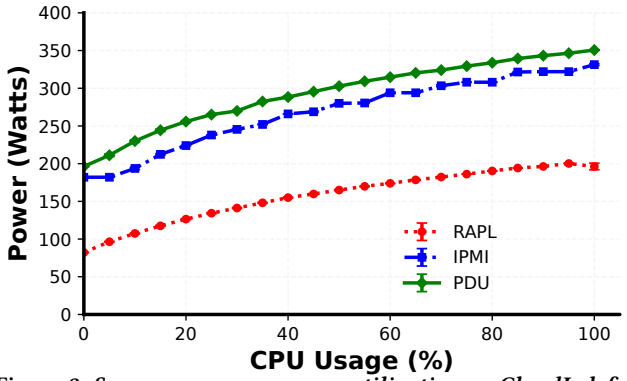

Figure 3: ***Server power versus core utilization on CloudLab for sensors at different points in the power distribution system.***

4 Implementation

We implement a prototype of CarbonShare in Python 3.7+, directly extended CarbonContainers [40], which consists of microservices architecture consisting of multiple daemon services that communicate via gRPCs, monitor per-application power and carbon emissions, and enforce a configurable cluster-wide carbon budget and assigned per-application carbon weights. The enforcement policy is also configurable based on the policies described in §3. The current prototype implementation runs on Ubuntu 22.04 and uses LXD (5.21.5 LTS) [30] with clustering enabled as its underlying container orchestration platform, such that LXD manages the placement of newly created containers within the cluster. Our prototype interfaces directly with our CarbonShare testbed for CloudLab [18], enabling it to get real-time power data from CloudLab servers' connected baseboard management controllers (BMCs) and power distribution units (PDUs).

This integration above allows our prototype to access accurate power usage measurements for each server in the cluster. Our prototype has two primary microservices, i) a monitoring module that measures container-level resource utilization and ii) an enforcement module that receives data from the monitor and makes enforcement decisions based on the configured policy. We discuss each service's implementation below, and then present our prototype's integration with CloudLab. We plan to publicly release CarbonShare on GitHub under a permissive open-source license, and provide public access to our CarbonShare testbed for CloudLab to enable further research on energy- and carbon-efficiency in cloud computing. Conducting such research is challenging because both public and private cloud platforms generally do not provide access to back-end power data.

Policy Enforcement Service. The enforcement service runs on a single "head" node, and is responsible for making control decisions for each container based on the system's carbon enforcement policy. At a configurable time interval, e.g., every second, the enforcement service queries the monitoring service running on each worker node and receives resource utilization measurements for all LXD containers running in the system. The lower this monitoring interval, the more accurately the system can capture containers' real-time power usage. The service also queries the power monitoring interface and its carbon data source at this time to receive power readings from the PDU of each worker node, which combined with the resource utilization measurements enable it to attribute energy consumption to each container.

Our CarbonShare prototype interfaces with the electricityMaps [5] API to determine the carbon-intensity of the electric grid supplying power to each server based on its location. Given the energy consumption $e(t)$ of a container over the interval t and the carbon intensity $c(t)$ over this same interval, CarbonShare computes the container's carbon emissions total $C(t) = e(t) \cdot c(t)$. The enforcement service's actions then depend on the policy. For the footprint-fair policy, the service updates each container's total emissions for the current budgeting period, and suspends the container if its fair-share is reached. In the rate-fair case, the service computes the maximum number of cores (or fractions of a core) that will keep each container beneath its fair-rate for the next interval, and applies the appropriate resource quota using LXD's container resource control mechanisms to prevent exceeding this maximum.

Monitoring Service. CarbonShare monitors each container's resource utilization to properly attribute energy usage and carbon emissions. To do so, a monitoring service runs on each worker node in the cluster, and exposes an API for the enforcement service above to query resource utilization, e.g., via `cgtop` which reports utilization by cgroup. As mentioned above, the enforcement service maps this utilization to power usage based on the server's aggregate power and a power model. Our system can support fine-grained carbon measurements for little overhead in terms of power and compute. Querying the PDU is performed through a simple REST API call, which requires minimal overhead. The cgtop system tool also incurs negligible overhead, with $\leq$ 1% CPU utilization of a 40-core machine resulting in $< 1W$ of power draw.

Testbed Integration. As mentioned above, our CarbonShare prototype integrates with our CarbonShare testbed for CloudLab that provides real-time power metrics from each server's PDU and BMC. The testbed stores these metrics in a Prometheus time-series database and makes them accessible via a REST API and visually via a Grafana dashboard. The PDU data includes server power draw, voltage, and current, while the BMC data includes the same power metrics in addition to server-specific cooling metrics, including fan speed and temperature. The REST API updates these readings every 15 seconds. Since our CarbonShare prototype focuses on the power data, we examined the relative accuracy of server-level power data from both the PDU and the BMC, which supports the IPMI protocol. The CloudLab servers also support RAPL, which is Intel's system for monitoring server power that is exposed by the OS. Note that RAPL is not available for non-Intel processors, and is thus not a general method for monitoring server power. Specifically, for these experiments, we ran the stress-ng tool with a range of target utilizations and took power readings from each sensor.

Figure 3 shows the results. In particular, the figure shows that power measurement becomes increasingly less accurate as the sensor moves closer to the server, as closer sensors ignore power consumption between their position and the PDU. The figure also confirms, as mentioned previously, that server power draw is roughly linear with core utilization. In this case, RAPL does not capture the power draw of the many components not connected to the CPU socket, such as the cooling system, i.e., fan, power supply, BMC, etc., which account for roughly 50% of the power usage. Likewise, the BMC/IPMI power readings do not account for its own power, which results in a smaller discrepancy between its measurements and those taken from the PDU. As a result, CarbonShare uses PDU measurements to take full account of a server's power.

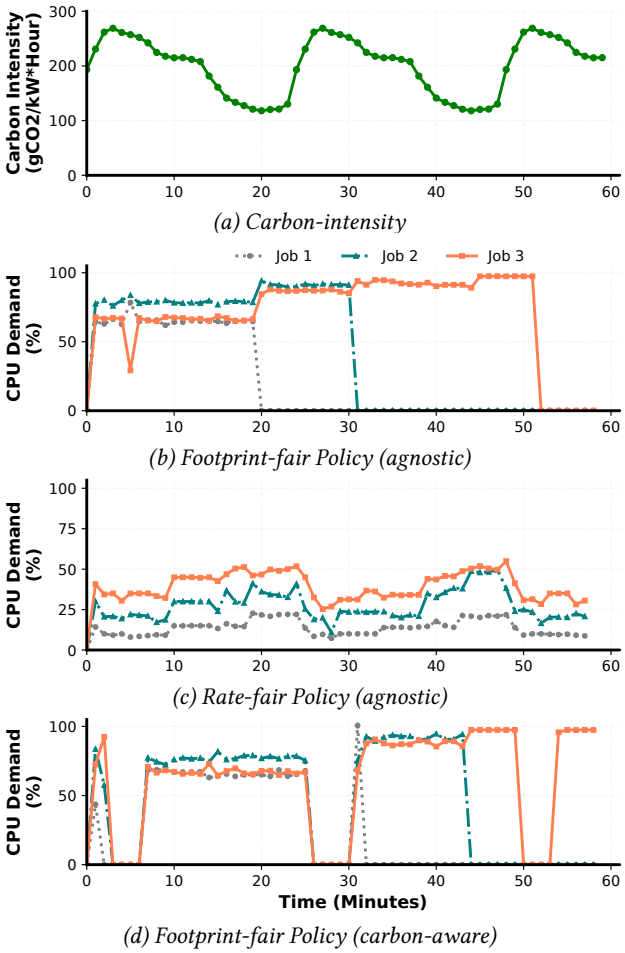

Figure 4: *CarbonShare prototype enforcing a carbon limit on three batch jobs with a carbon weight ratio of 1:2:3 under different types of enforcement and job policies.*

5 Experimental Evaluation

Below, we demonstrate our prototype's basic functions at a small-scale for different policy and weight settings for batch and interactive jobs (§5.1). We then conduct a large-scale evaluation in simulation to quantify performance across a large set of representative jobs from a production industry job trace (§5.2).

5.1 Prototype Evaluation

5.1.1 Batch Jobs. Figure 4 illustrates CarbonShare's enforcement policies across three identical batch jobs in a cluster. In this experiment, we assign carbon weights in a 1:2:3 ratio for jobs 1, 2, and 3, respectively. In this case, the jobs are computationally-intensive HPC jobs. Thus, job 1 receives a 16.7% carbon share, job 2 receives a 33.3% carbon share, and job 3 receives a 50% carbon share. The cluster size for this experiment is 3 homogeneous CloudLab servers (rs630 class), where each job runs on its own server. While the jobs

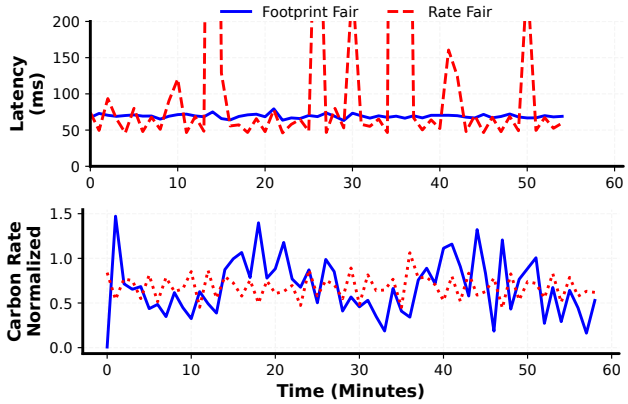

Figure 5: ***Interactive service latency (a) and normalized carbon rate (b) under a footprint-fair and rate-fair policies.***

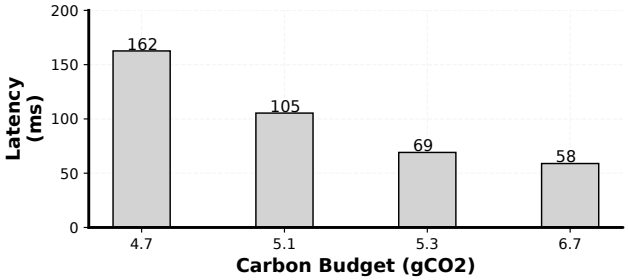

Figure 6: ***Interactive latency for jobs with different carbon budgets. Higher budgets enable lower latency SLOs.***

could run co-located on a server, we isolate these jobs to independent servers to isolate the specific effect of carbon enforcement on their performance. Figure 4(a) shows energy's variable carbon-intensity for the experiment. Since carbon-intensity changes slowly, our experiment emulates much longer jobs over an hour-long period. That is, we speed up a 72-hour carbon-intensity trace from California to one hour to facilitate our experiments, and set our interval for budgeting carbon emissions to 1 hour. Figures 4(b), (c), and (d) then show the performance of each job, as a function of core utilization, under multiple different carbon-fair enforcement policies and job behaviors.

Figure 4(b) illustrates jobs using a footprint-fair policy but that are agnostic, i.e., not responsive, to their carbon emissions. In this case, the jobs run unbounded as normal as if there is no limit on their carbon emissions. The behavior of these agnostic jobs running under a footprint fair budget reflect how we expect real-world jobs to behave currently, except jobs would not suspend since such budgeting policies are currently not implemented. Of course, in this case, once jobs exhaust their carbon allocation, CarbonShare suspends their execution until the next budgeting period. As expected, CarbonShare suspends jobs in order of, and in proportion to, their allocated carbon share with job 1 being suspended first followed by job 2 and then job 3. Figure 4(c) next illustrates the same jobs executing under the rate-fair policy. In this case, CarbonShare computes and enforces their fair emissions rate, and then continuously adjusts their resource allocation to satisfy this rate. As a result, none of the jobs exhaust their carbon allocation and need to be suspended. Instead, their core utilization varies with energy's carbon-intensity in (a) such that when carbon-intensity decreases their core utilization increases, e.g., around minutes 20 and 35, and vice versa. In this case, the core utilization for each job varies with carbon-intensity, and is proportional to their weight.

Notably, the total amount of computation performed by the rate-fair policy is more than the footprint-fair policy in (b) where the jobs are not responsive to energy's carbon emissions. This occurs because in (b) the jobs operate at high utilization during high carbon periods at the outset and deplete their carbon allocation for much less overall computation, while in (c) the rate-fair policy continuously varies the rate and thus automatically limits core utilization during high carbon periods. This implicitly shifts jobs' demand at the outset to lower carbon periods later on. Figure 4(d) shows the jobs operating under the footprint-fair policy with carbon-aware policy that is responsive to variations in energy's carbon-intensity unlike in (b). The carbon-aware policy here is similar to prior work [43] that only uses resources when carbon is below some low threshold, and uses no carbon when it is above that threshold. Such carbon-aware jobs operating under the footprint-fair policy have the most flexibility to shift their load with energy's carbon-intensity. As a result, this approach results in more computation than the rate-fair approach, as it maximizes computation performed during the lowest carbon times.

5.1.2 Interactive Services. We also illustrate CarbonShare in managing interactive services, in this case a web service. For this experiment, we assume the same variable carbon-intensity as in Figure 4(a), and use the MediaWiki [8] site hosting a replica of the German Wikipedia. We use a workload generator to send a steady rate of requests to the web site [6]. We run the application under both the rate-fair and footprint-fair policies, and define a latency SLO for the site of 75ms. Figure 5(top) shows average request latency for both the site operating under the footprint-fair and rate-fair policies, while Figure 5(bottom) shows the normalized carbon rate. As shown, the rate-fair policy maintains a near-constant carbon rate at or below 1, which represents its fair carbon rate that evenly spreads its carbon allocation over its budgeting interval. However, in doing so, whenever energy's carbon-intensity increases, CarbonShare decreases the service's allocated resources to compensate, which results in a significant increase in latency well beyond the SLO, with 18% of requests missing the target. In contrast, the footprint-fair policy enables the service to more effectively plan its resource and carbon usage. Specifically, the carbon rate is below the fair-rate when carbon-intensity is low, which enables the service to "save carbon" that it can use when energy's carbon-intensity increases. As a result, the footprint-fair policy is able to maintain a consistent latency at or below the SLO despite significant variations in energy's carbon-intensity.

To demonstrate the effect of a carbon budget on interactive service performance, we repeat this experiment, allocating different amounts of carbon budget to the interactive service. Figure 6 shows the average request latency under a number of carbon budgets. As depicted, when the interactive service is given a larger budget to operate, it is able to consistently support lower latencies, as expected.

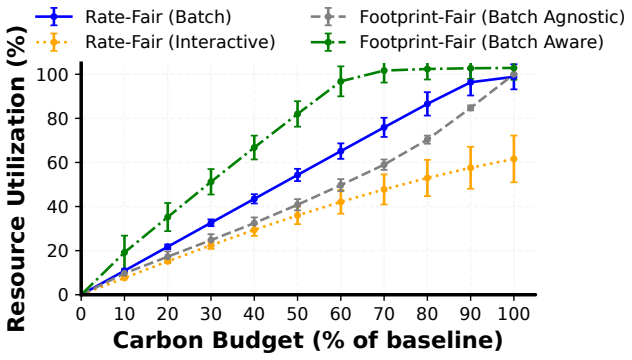

Figure 7: ***Average per-job resource utilization as a function of the carbon budget for different policies.***

Notably, the relationship between budget and performance is not strictly linear. Overprovisioning carbon budgets to an interactive job will naturally have diminishing returns on latency, as network constraints will become the bottleneck instead of computation as the budget increases. On the other hand, underprovisining such jobs can potentially cause exponentially worse latencies if a service is not able to process requests faster than they arrive due to computing constraints. Given the significant risks associated with this underprovisioning, cluster operators may elect to give larger weights to interactive jobs that must share budgets with batch jobs to avoid SLO violations.

5.2 Large-scale Evaluation

We next perform a large-scale evaluation in simulation over many jobs, different job types, enforcement policies, regions, weight assignments, and forecasting uncertainty to understand how CarbonShare's performance differs at scale. For these experiments, we select a random sample of 1000 jobs from a public job trace released by Azure [1]. Specifically, we utilize the Azure VM traces dataset, which report the minimum, maximum, average, and p95 average of CPU utilization per VM over 5 minute intervals. The traces capture jobs' variable resource usage, e.g., core utilization and memory, over their running time. The simulator replays these jobs' utilization, simulating their energy and carbon usage over time while applying a configurable carbon enforcement policy. Carbon data for these simulations are taken from ElectricityMaps. We take a representative day-long trace from the California ISO region, and replay the carbon intensity trace concurrently to the jobs. We model the energy profile of these simulated servers on the CloudLab servers used to run the prototype demonstrations above. By modeling each job's energy usage in this way, we can translate this into carbon emissions by simply multiplying this value by the carbon-intensity at the corresponding simulated time in the carbon trace. For these simulations, we assume that jobs run concurrently over 24 hours with adequate resources to support them.

5.2.1 Evaluation of Enforcement Policies. We first evaluate how different enforcement policies and job behaviors impact job performance at scale for the same representative carbon-intensity trace from Figure 4(a). Again, we compare the rate-fair and footprint-fair policies, as well as batch and interactive variants of jobs. We set the weights of the jobs such that they are one-to-one with their unbounded core-time utilization, e.g., carbon budgets are directly proportional to the total amount of computation required by each job. As mentioned in Section 2, these values are equivalent to their Shapley values when maximizing core-time is the objective.

Our simulator assumes if the jobs are batch they can shift their demand to the future (if they are throttled due to high carbon), while if they are interactive they cannot shift demand but must implicitly shed load if throttled. For the footprint-fair policy with batch jobs, we also compare jobs that are carbon-agnostic, in that they do not change their behavior to respond to changes in energy's carbon-intensity, and carbon-aware, which prioritize using their carbon allocation during low carbon periods to maximize their carbon-efficiency. For this simulation, we assume that jobs have access to an accurate forecast of carbon-intensity, enabling them to effectively plan their carbon spending. We relax this assumption in Section 5.2.4 to quantify the impact of prediction error.

Figure 7 plots the average resource utilization of these enforcement policy and job combinations as a percentage of jobs' carbon budget, where the error bars represent the standard deviation. Here, we normalize the carbon budget relative to the carbon emissions from running the jobs without any limitations. As expected, as the carbon budget decreases, jobs' resource utilization also decreases, since CarbonShare throttles resource usage to enforce the carbon budget. The highest resource utilization is the footprint-fair policy for carbon-aware batch jobs, as these jobs shift demand to low-carbon periods to maximize their carbon-efficiency. Batch jobs under the rate-fair policy achieve the next highest resource utilization, since, as discussed earlier, the rate-fair policy automatically does some limited shifting of resource usage to lower carbon periods. As a result, the rate-fair policy performs better than the footprint-fair carbon-agnostic batch jobs that do not attempt to align with low-carbon periods. In contrast, interactive jobs using the rate-fair policy perform the worst, as they have no ability to shift their load, and during periods of low demand often operate below their fair rate. Unfortunately, in this case, during periods of high demand or carbon-intensity they cannot operate above their fair rate to compensate, which lowers their overall resource usage.

Figure 8 shows the corresponding carbon emissions from the same experiment from the same policy and job behavior combinations. Figure 8(top) shows that the footprint-fair policy with a carbon-aware job not only maximizes resource usage, but also results in the lowest carbon emissions. The combination of high resource utilization and low carbon emissions results in an even greater discrepancy in carbon-efficiency, which is work done per mass of carbon emitted. Figure 8(bottom) plots the carbon-efficiency and shows how the combination of both metrics increases the difference between the carbon-aware jobs using the footprint-fair policy versus the other policies and job behaviors.

5.2.2 Impact of Carbon Weight Assignment. CarbonShare throttles applications when their carbon emissions reach a specific rate (under the rate-fair policy) or an aggregate amount (under the footprint-fair policy). Thus, the assignment of weights to applications is important in maximizing the utilization of the cluster, especially for the carbon-conserving footprint-fair policy. For example, if an administrator assigns a large carbon share to a user or job

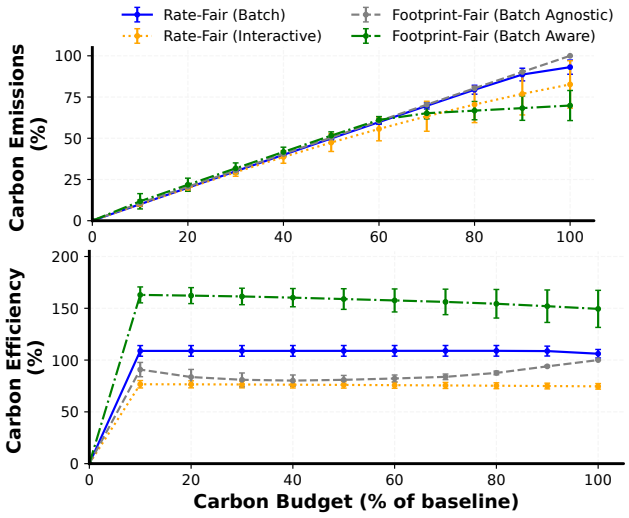

Figure 8: ***Average carbon emissions (top) and carbon-efficiency (bottom) for 1000 random batch jobs under different policies as a function of normalized carbon allocation.***

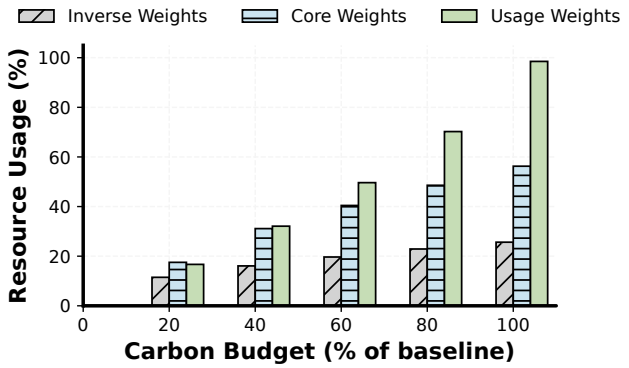

Figure 9: ***Cluster utilization as a function of allocated carbon budget for different weight assignment strategies under the footprint-fair policy.***

that does not consume many resources, then much of the share will go unused since, by default, CarbonShare is carbon-conserving and not work-conserving. This may be the intended result, as applications may choose to reduce their usage to reduce carbon emissions.

To quantify the impact of weight assignment on performance, we experimented with different weight assignment policies for the same 1000 jobs as the previous experiment. Figure 9 shows the results, which plots the carbon budget on the x-axis and the aggregate cluster utilization on the y-axis for three different carbon weight assignment policies. The policies are *inverse weights*—allocate carbon weights inversely proportional to resource usage—*core weights*—allocate carbon weights proportional to jobs' requested number of cores—and *usage weights*—allocate carbon weights proportionately to resource usage. As expected, allocating carbon weight inversely proportional to resource usage results in the lowest utilization, as jobs with low resource usage receive a large share of the carbon budget that ultimately goes unused. In contrast, allocating carbon

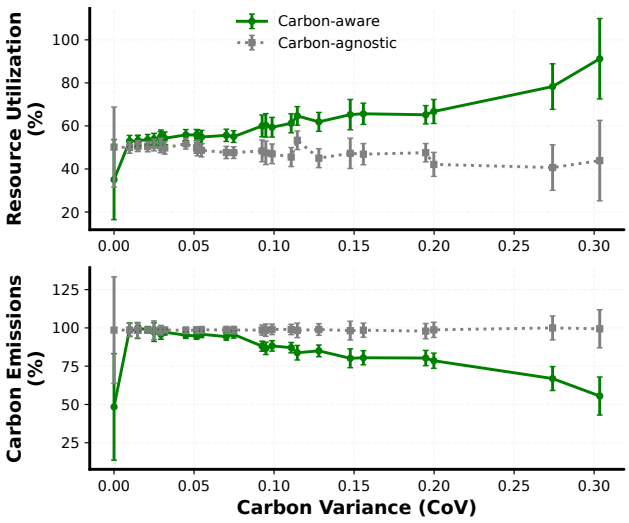

Figure 10: ***Resource utilization for a 50% carbon budget (top) and emissions for a 100% carbon budget (bottom) across 28 regions as their variation in carbon-intensity changes.***

weights proportionately to resource usage increases utilization such that it is close to the carbon budget percentage, which is ideal. The core weights policy falls somewhere in the middle as it is a rough approximation of the usage weight policy.

5.2.3 Cross-Regional Analysis. The benefits of jobs being responsive to variations in carbon-intensity under CarbonShare, of course, depend on the magnitude and frequency of these variations. Specifically, shifting resource usage to align with low-carbon periods is more important in regions with larger and more frequent variations in carbon-intensity. To understand this effect, we again simulated the same 1000 jobs from before across 28 different regions with different carbon-intensity traces. In this case, we run the jobs using the footprint-fair policy when both carbon-agnostic and carbon-aware with a 50% carbon budget, i.e., the carbon limit is set to 50% of the carbon emissions when the job is run without any throttling. Figure 10(top) shows resource utilization as a function of the coefficient of variation (CoV) in carbon-intensity for the region. A higher coefficient of variation, which is the mean divided by the standard deviation, indicates more and larger variations in carbon-intensity. As expected, as carbon variations increase there is a greater discrepancy between the carbon-aware and carbon-agnostic policies, indicating the increasing importance of being carbon-aware.

We also run the same experiment but where the carbon budget is set to 100% of carbon-agnostic jobs to quantify how carbon emissions changes with the variation in carbon-intensity. Figure 10(bottom) shows that, similarly, as the variation in carbon increases, the discrepancy in carbon emissions also increases between the carbon-agnostic and carbon-aware jobs. In particular, the carbon-aware jobs use significantly less carbon in high variation regions because they shift their resource usage to better align with low-carbon periods. In contrast, when variations in carbon-intensity are low there is little benefit to being carbon-aware and shifting load to (slightly) lower carbon periods. Ultimately, if carbon-intensity never changes, then CarbonShare reverts to fair energy sharing.

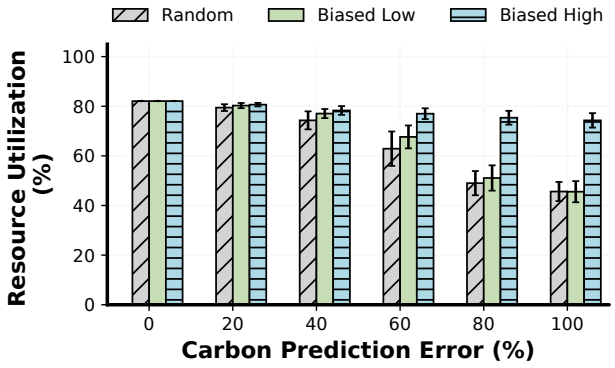

Figure 11: ***Average resource utilization for a 50% carbon budget with varying degrees of forecast error.***

5.2.4 Effect of Carbon Forecast Error. The primary drawback of the footprint-fair policy is that it requires jobs to carefully plan their resource usage over time based on predictions of future carbon-intensity. Our earlier experiments assumed perfect carbon-intensity forecasts. To evaluate the impact of forecast error, we vary the level of error in the 24-hour carbon-intensity forecast that each job bases its load shifting decisions on. We run these experiments for 1000 jobs as before with the footprint-fair policy and carbon-aware job behavior. Figure 11 shows the results for different error distributions that are i) uniformly random error, ii) error biased to predict lower values than the actual, and iii) error biased to predict higher values than the actual. The graph varies the percentage error on the x-axis and plots the average resource utilization on the y-axis. In this case, a lower utilization derives from jobs running during higher carbon periods that deplete their carbon budget for less resource usage.

As expected, as the forecast error increases the average resource utilization decreases as the jobs' ability to accurately plan their resource usage also decreases. However, importantly, there is not a significant decreases in resource utilization until the forecast error exceeds 40%. Even at 40% error in carbon forecasts, the drop in resource utilization is only 5-10% across the different error distributions. Prior work on carbon forecasting has shown that typical forecast errors are much less than 40% [31]. The underlying reason for this result is that to effectively align resource usage with periods of low carbon, jobs need only accurately identify high carbon and low carbon periods, rather than quantify the precise carbon-intensity. These periods can be identified even under high forecast errors, especially since energy's carbon-intensity exhibits strong diurnal patterns. In addition, the error distribution that biases the errors high tends to perform better than uniformly random or biasing low because low predictions errors have a floor of 0, which can eliminate the distinction between a high and low period. For example, if errors are biased low, then a period with 0 carbon-intensity may be forecasted to be 0, while a period with higher carbon-intensity might also be forecasted as 0.

6 Related Work

CarbonShare intersects a range of prior work in managing resources, energy, and carbon in shared clusters, summarized below.

Fair-share Scheduling. Our approach is inspired by foundational work on early fair-share schedulers [26] and policies, such as WFQ [15, 26] and lottery scheduling [42]. Modern fair-share schedulers, such as DRF [21], extended such schedulers to fairly allocating multiple resource types, e.g., cores and memory, across a cluster. CarbonShare is orthogonal to prior work on fair-share resource scheduling, as it builds on an existing resource scheduling policy, such as DRF, by limiting the usage of resources to enforce a carbon budget. Similarly, recent work has adapted fair-share scheduling to fairly allocate energy to applications [19, 20, 29]. CarbonShare differs from this work in that it focuses on carbon, and not energy, which considers both energy's variable carbon-intensity and jobs' energy-efficiency. Since energy's carbon-intensity varies over time, jobs' responsiveness to these carbon variations is important.

Resource and Energy Management. CarbonShare builds on container orchestration platforms [7, 14, 41] that allocate resources to jobs in the form of containers [9]. CarbonShare also uses resource quotas to enforce its policies, similar to many existing systems for power management [27, 28]. However, prior work is largely motivated by constraints on maximum power capacity, while we use these mechanisms to avoid exceeding a carbon emissions budget.

Carbon Attribution. Recent work on FairCO2 [24] provides a system that fairly attributes not only operational, but also embodied, carbon to user-level processes/containers in a multi-user datacenter environment. CarbonShare differs in its focus on fairly enforcing carbon limits across users, while FairCO2 focuses on how to fairly attribute carbon (operational and embodied) to users.

Carbon Management. There has been significant recent work on improving datacenter carbon-efficiency [12, 16, 22, 23, 25, 33, 38, 43, 45]. Much of this work focuses on optimizing jobs' carbon-efficiency by exploiting variations in energy's carbon-intensity. Our work enables such jobs to run in a shared cluster that adheres to a limited carbon budget. Perhaps most related to CarbonShare is prior work on Carbon Containers [40], which focuses on rate-limiting the carbon emissions of a single resource container. Instead, CarbonShare focuses on fairly allocating a carbon budget (not rate) across many jobs and containers within a cluster.

CarbonShare is also related to prior work on an ecovisor [38], which virtualizes the energy system for a datacenter and exposes it programmatically to applications. In contrast, CarbonShare operates at a higher level of abstraction—above the resource scheduler rather than beneath it—and focuses on the policy by which applications share the burden of limited carbon. Instead, ecovisor focuses on virtualization mechanisms and explicitly does not define policies for sharing energy or resources.

7 Conclusion

This paper introduces the notion of carbon fairness for clusters that must adhere to a targeted amount of aggregate carbon emissions over some time interval. We design, implement, and evaluate CarbonShare, a system that enforces a cluster-wide limit on carbon emissions and design multiple carbon-fair policies for allocating limited carbon emissions to applications. We implement a CarbonShare prototype in LXD and evaluate it by extending CloudLab with power monitoring functions. Our results show that a footprint-fair policy improves performance and emits less carbon than a stricter

rate-limiting policy for carbon-aware jobs that are able to shift their usage to align with variations in energy's carbon-intensity.

Acknowledgements. This research is supported, in part, by NSF grants 2213636, 2325956, and 2105494

References

[1] 2024. AzurePublicDataset. https://github.com/Azure/AzurePublicDataset.

[2] 2024. Forbes, AI Is Pushing the World Toward an Energy Crisis. https://www.forbes.com/sites/arielcohen/2024/05/23/ai-is-pushing-the-world-towards-an-energy-crisis/.

[3] 2024. Goldman Sachs, AI is Poised to Drive 160% Increase in Data Center Power Demand. https://www.goldmansachs.com/intelligence/pages/AI-poised-to-drive-160-increase-in-power-demand.html.

[4] 2025. Center for Climate and Energy Solutions, State Climate Policy Maps. https://www.c2es.org/content/state-climate-policy/.

[5] 2025. Electricity Maps. https://app.electricitymaps.com/map/72h/hourly.

[6] 2025. httpmon. https://github.com/cloud-control/httpmon/tree/master.

[7] 2025. Kubernetes: Production-grade Container Orchestration. https://kubernetes.io/.

[8] 2025. MediaWiki. https://www.mediawiki.org/wiki/MediaWiki.

[9] Gaurav Banga, Peter Druschel, and Jeffrey Mogul. 1999. Resource Containers: A New Facility for Resource Management in Server Systems. In *OSDI* (1999).

[10] Zach Bright. 2023. EnergyWire, Southern Co. may postpone coal plant shutdowns. https://subscriber.politicopro.com/article/eenews/2023/11/03/southern-co-may-postpone-coal-plant-shutdowns-00125067.

[11] Justine Calma. 2024. The Verge, Microsoft's AI obsession is jeopardizing its climate ambitions. https://www.theverge.com/2024/5/15/24157496/microsoft-ai-carbon-footprint-greenhouse-gas-emissions-grow-climate-pledge.

[12] Andrew Chien. 2021. Driving the Cloud to True Zero Carbon. *CACM* 64, 2 (January 2021).

[13] Maxime Colmant, Pascal Felber, Romain Rouvoy, and Lionel Seinturier. 2017. WattsKit: Software-Defined Power Monitoring of Distributed Systems. In *CCGRID*.

[14] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017-10-14).

[15] Alan Demers, Srinivasan Keshav, and Scott Shenker. 1989. Analysis and Simulation of a Fair Queueing Algorithm. *ACM SIGCOMM Computer Communication Review* 19, 4 (August 1989).

[16] Jesse Dodge, Taylor Prewitt, Remi Tachet des Combes, Erika Odmark, Roy Schwartz, Emma Strubell, Alexandra Sasha Luccioni, Noah A. Smith, Nicole DeCario, and Will Buchanan. 2022. Measuring the Carbon Intensity of AI in Cloud Instances. In *2022 ACM Conference on Fairness, Accountability, and Transparency*.

[17] Stanley Dunlap. 2024. Georgia Recorder, State utility regulators approve Georgia Power plan to use fossil fuels to power data centers. https://georgiarecorder.com/2024/04/16/state-utility-regulators-approve-georgia-power-plan-to-use-fossil-fuels-to-power-data-centers/.

[18] Dmitry Dupyakin, Robert Ghiselli Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh B. Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Lawrence Hugh Landweber, Chip Elliot, Michael Zink, Emmanel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *USENIX Annual Technical Conference (ATC)*.

[19] Pierre-François Dutot, Yiannis Georgiou, David Glesser, Laurent Lefevre, Millian Poquet, and Issam Rais. 2017. Towards Energy Budget Control in HPC. In *CCGRID*.

[20] Yiannis Georgiou, David Glesser, Krzysztof Rzadca, and Denis Trystram. 2015. A Scheduler-Level Incentive Mechanism for Energy Efficiency in HPC. In *CCGRID*.

[21] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *NSDI*.

[22] Inigo Goiri, Kien Le, Md. Ehtesamul Haque, Ryan Beauchea, Thu D. Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. 2011. GreenSlot: Scheduling Energy Consumption in Green Datacenters. In *Supercomputing (SC)*.

[23] Udit Gupta, Young Geun Kim, Sylvia Lee, Jordan Tse, Hsien-Hsin S. Lee, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. 2021. Chasing Carbon: The Elusive Environmental Footprint of Computing. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*.

[24] Leo Han, Jash Kakadia, Benjamin C. Lee, and Udit Gupta. 2025. Fair-CO2: Fair Attribution for Cloud Carbon Emissions. In *ISCA*.

[25] Walid Hanafy, Qianlin Liang, Noman Bashir, David Irwin, and Prashant Shenoy. 2024. CarbonScaler: Leveraging Cloud Workload Elasticity for Optimizing Carbon-Efficiency. In *SIGMETRICS/Performance*.

[26] Judy Kay and Piers Lauder. 1988. A Fair Share Scheduler. *CACM* 31, 1 (January 1988).

[27] Vasileios Kontorinis, Liuyi Eric Zhang, Baris Aksanli, Jack Sampson, Houman Homayoun, Eddie Pettis, Dean M. Tullsen, and Tajana Simunic Rosing. 2012. Managing Distributed UPS Energy for Effective Power Capping in Data Centers. In *ISCA*.

[28] Shaohong Li, Xi Wang, Faria Kalim, Xiao Zhang, Sangeetha Abdu Jyothi, Karan Grover, Vasileios Kontorinis, Nina Narodytska, Owolabi Legunsen, Sreekumar Kodakara, et al. 2020. Thunderbolt: Throughput-Optimized, Quality-of-Service-Aware Power Capping at Scale. In *(OSDI)*.

[29] Qianlin Liang, Walid A. Hanafy, Noman Bashir, David Irwin, and Prashant Shenoy. 2023. Energy Time Fairness: Balancing Fair Allocation of Energy and Time for GPU Workloads. In *2023 IEEE/ACM Symposium on Edge Computing (SEC)*.

[30] Canonical Ltd. 2025. LXD. https://documentation.ubuntu.com/lxd/en/latest/.

[31] Diptyaroop Maji, Prashant Shenoy, and Ramesh K. Sitaraman. 2022. CarbonCast: Multi-Day Forecasting of Grid Carbon Intensity. In *Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*.

[32] Naureen S Malik, Mark Chediak, and Ari Natter. 2024. Bloomberg, FirstEnergy Scraps 2030 Climate Goal in Rare Embrace of Coal. https://www.bloomberg.com/news/articles/2024-02-09/firstenergy-scraps-2030-climate-goal-in-rare-embrace-of-coal.

[33] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. *Carbon Emissions and Large Neural Network Training*. Technical Report. arXiv.

[34] Andreas Schmidt, Gregory Stock, Robin Ohs, Luis Gerhorst, Benedict Herzog, and Timo Honig. 2023. carbond: An Operating System Daemon for Carbon Awareness. In *HotCarbon*.

[35] Joe Schulz. 2023. Wisconsin Public Radio, We Energies doesn't rule out further delays to coal plant retirements during earnings call. https://www.wpr.org/economy/we-energies-doesnt-rule-out-further-delays-coal-plant-retirements-during-earnings-call.

[36] Lloyd S. Shapley. 1951. Notes on the N-Person Game — II: The Value of an N-Person Game. *RAND Research Memorandum* (August 1951).

[37] Kai Shen, Arrvindh Shriraman, Sandhya Dwarkadas, Xiao Zhang, and Zhuan Chen. 2013. Power Containers: An OS Facility for Fine-Grained Power and Energy Management on Multicore Servers. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*.

[38] Abel Souza, Noman Bashir, Jorge Murillo, Walid Hanafy, Qianlin Liang, David Irwin, and Prashant Shenoy. 2023. Ecovisor: A Virtual Energy System for Carbon-Efficient Applications. In *ASPLOS*.

[39] Thanathorn Sukprasert, Abel Souza, Noman Bashir, David Irwin, , and Prashant Shenoy. 2024. On the Limitations of Carbon-Aware Temporal and Spatial Workload Shifting in the Cloud. In *EuroSys*.

[40] John Thiede, Noman Bashir, David Irwin, and Prashant Shenoy. 2023. Carbon Containers: A System-level Facility for Managing Application-level Carbon Emissions. In *SoCC*.

[41] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-Scale Cluster Management at Google with Borg. In *EuroSys*.

[42] Carl A Waldspurger and William E Weihl. 1994. Lottery Scheduling: Flexible Proportional-Share Resource Management. In *OSDI*.

[43] Philipp Wiesner, Ilja Behnke, Dominik Scheinert, Kordian Gontarska, and Lauritz Thamsen. 2021. Let's Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions. In *Middleware*.

[44] John D. Wilson and Zach Zimmerman. 2023. The Era of Flat Power Demand is Over. https://gridstrategiesllc.com/wp-content/uploads/2023/12/National-Load-Growth-Report-2023.pdf.

[45] Jiali Xing, Bilge Acun, Aditya Sundarrajan, David Brooks, Manoj Chakkaravarthy, Nikky Avila, Carole-Jean Wu, and Benjamin C. Lee. 2023. Carbon Responder: Coordinating Demand Response for the Datacenter Fleet. arXiv:2311.08589 [cs.DC]